Metalcraft, Inc.

RFID Label Read/Write System
Developer's Guide

Version 1.0

Revision History

Date	Version	Description	Author
April 18, 2002	1.0	Created document	Timothy A. Herrick

©Metalcraft, Inc., 2002

Table of Contents

Preface 5

1. A Short History 5
2. Documentation Resources 5
3. Terminology and Notation 5

Add Support for a New RFID Device 6

1.1 CRFIDDevice 6
1.2 Overriding functions 6
1.3 The Constructor 7
1.4 User Interface 7

2. Add Support for a New Barcode Device 8

2.1 CBarcodeDevice 8
2.2 Overriding functions 8
2.3 The Constructor 8
2.4 User Interface 9

3. User Interface and Processing Thread Synchronization 10

3.1 Start and Stop 10
3.2 Exiting the Application 11
3.3 Thread Died 11

4. Processing Thread and Web Handler Synchronization 12

5. Port Wiring 13

6. File Formats 14

6.1 RFID Image Database 14
6.2 Save Settings 15

Table of Figures

Figure 3-1 Start and Stop....10
Figure 3-2 Exiting the Application....11
Figure 3-3 Thread Died....11
Figure 4-1 Processing Thread and Web Handler Signals....12
Figure 5-1 Port Wiring Diagram....13

Preface

1. A Short History

The Software Processing and Control Subsystem (SPCS) and the RFID Image Database Tool were originally developed by Iowa State's May02-01 senior design team. These combined with Metalcraft's transport subsystem, form Metalcraft's RFID Label Read/Write and Transport System.

2. Documentation Resources

1. RFID Label Read/Write System Developer's Guide, DEVG, V1.0, 2002
2. Software Processing and Control Subsystem User's Manual, SPCSUM, V1.0, 2002
3. RFID Image Database Tool User's Guide, DTUG, V4.0, 2002

3. Terminology and Notation

RFID Image

A record containing the binary data to be programmed into RFID tags, a description of where in the tag to put it and the associated barcode (if one exists).

RFID Image Database

A collection of RFID Images

Schema

Description of RFID Image structure

Software Processing and Control System (SPCS)

The software system that controls the devices and web handler that actually writes the RFID labels.

Add Support for a New RFID Device

1.1 CRFIDDevice

The abstract class CRFIDDevice defines the interface used to control a RFID device. When adding support for a new RFID device you must derive your class from CRFIDDevice.

1.2 Overriding functions

The following functions must be overridden in your derived class.

1.2.1 void DisplayPropWnd()

When this function is called, a window should display which prompts the user to select settings. Most RFID devices will at least allow the user to select which port the device is on. You will need to create the window resources and any associated classes for this window. If your device has no user selectable settings, you should display a message box stating that it doesn't.

1.2.2 void SaveSettings(CFile* saveFile)

SPCS, barcode device, and RFID device settings are all save into a single file. This function is called as part of the SPCS save settings operation. The file object will already be opened. Simply write your settings to the file. You must not explicitly move the file pointer and you must not close the file. These two operations can corrupt the entire saved settings file. See section 5.2 Save SettingsSave Settings

1.2.3 void LoadSettings(CFile* loadFile)

This function loads settings from the saved settings file. You must read back the data exactly like you stored it in the SaveSettings function, no more, no less. The calling procedure assumes that the file pointer is positioned immediately after your portion of the file when this function returns. See section 5.2 Save Settings

1.2.4 bool WriteAndVerifyBlock(WORD block, BYTE* buffer, WORD size)

This function should write a block of data to the current RFID tag. Size indicates the size of the buffer and block indicates which block to write the data into. This function must also verify the data was written correctly. It should return true if function was successful, otherwise it should return false. The calling procedure will handle the allocation and deallocation of buffer; so do not free this memory space.

1.2.5 WORD ReadBlock(WORD block, BYTE* buffer, WORD buffer_size)

This function should read a block of data from the current RFID tag. Size indicates the size of the buffer and block indicates which block to read the data from. Upon completion the function should return the actual number of bytes read. The calling procedure will handle the allocation and deallocation of buffer; so do not free this memory space and do not allocate space to buffer.

1.2.6 WORD GetBlockSize()

Returns the block size of the tag in bytes.

1.2.7 WORD GetBlockCount()

Returns the number of blocks on the tag.

1.2.8 CString GetErrorString()

Returns a description of the last reported error.

1.2.9 CString GetDeviceName()

Returns a short description of the device. This description is used in the RFID Device selection list as well as in the saved settings file. As a result, make it descriptive but don't make it excessively long.

1.2.10 bool Initialize()

This function must open a communications port to the device and properly configure the device. If the initialization is successful the member variable named "Active" should be set to true, otherwise it should be set to false. This function returns the value of the member variable named "Active".

1.3 The Constructor

The constructor should set the protected member variable named "Active" to false.

1.4 User Interface

In addition to creating a class to control your new device, you must also update certain user interface functions. These functions are located in DeviceFncs.cpp

1.4.1 void CSPCSDlg::LoadRFIDComboBox(CComboBox *pCombo)

This function adds a description to the RFID Device selection list for each supported device. Simply create an instance of your derived class and call the AddString function of the CComboBox object, passing it a description of your device. For example, let's assume your derived class is called CMyRFIDDevice. Simply add the following code to the function.

```
CMyRFIDDevice rfidDevice;
pCombo->AddString(rfidDevice.GetDeviceName());
```

1.4.2 void CSPCSDlg::OnSelchangeRfidCombo()

This function is called when a user selects a new RFID device from the RFID device selection list. Let's assume your derived class is called CMyRFIDDevice. Simply add the following code to the function.

```
CMyRFIDDevice * mrd = new CMyRFIDDevice (); // create an instance
if(str == mrd ->GetDeviceName())            // did the user select this device?
  APP->rfidDevice = mrd;                    // if yes assign it to the global variable
else
  delete mrd;                               // else delete the instance
```

1.4.3 devices.h

There is still one last thing to do to add support for a new RFID device. Open devices.h and include the .h file for your derived class.

2. Add Support for a New Barcode Device

2.1 CBarcodeDevice

The abstract class CBarcodeDevice defines the interface used to control a barcode device. When adding support for a new barcode device you must derive your class from CBarcodeDevice.

2.2 Overriding functions

The following functions must be overridden in your derived class.

2.2.1 void DisplayPropWnd()

When this function is called, a window should display which prompts the user to select settings. Most barcode devices will at least allow the user to select which port the device is on. You will need to create the window resources and any associated classes for this window. If your device has no user selectable settings, you should display a message box stating that it doesn't.

2.2.2 void SaveSettings(CFile* saveFile)

SPCS, barcode device, and RFID device settings are all save into a single file. This function is called as part of the SPCS save settings operation. The file object will already be opened. Simply write your settings to the file. You must not explicitly move the file pointer and you must not close the file. These two operations can corrupt the entire saved settings file. See section 5.2 Save SettingsSave Settings

2.2.3 void LoadSettings(CFile* loadFile)

This function loads settings from the saved settings file. You must read back the data exactly like you stored it in the SaveSettings function, no more, no less. The calling procedure assumes that the file pointer is positioned immediately after your portion of the file when this function returns. See section 5.2 Save Settings

2.2.4 bool ReadBarcode(CString& data)

This function should read data from the barcode reader. The data is stored as a string into "data". Upon completion the function should return true if the barcode was read and false if the barcode wasn't read.

2.2.5 CString GetErrorString()

Returns a description of the last reported error.

2.2.6 CString GetDeviceName()

Returns a short description of the device. This description is used in the Barcode Device selection list as well as in the saved settings file. As a result, make it descriptive but don't make it excessively long.

2.2.7 bool Initialize()

This function must open a communications port to the device and properly configure the device. If the initialization is successful the member variable named "Active" should be set to true, otherwise it should be set to false. This function returns the value of the member variable named "Active".

2.3 The Constructor

The constructor should set the protected member variable named "Active" to false.

2.4 User Interface

In addition to creating a class to control your new device, you must also update certain user interface functions. These functions are located in DeviceFncs.cpp

2.4.1 void CSPCSDlg:: LoadBarcodeComboBox (CComboBox *pCombo)

This function adds a description to the Barcode Device selection list for each supported device. Simply create an instance of your derived class and call the AddString function of the CComboBox object, passing it a description of your device. For example, let's assume your derived class is called CMyBarcodeDevice. Simply add the following code to the function.

```
CMyBarcodeDevice barcodeDevice;
pCombo->AddString(barcodeDevice.GetDeviceName());
```

2.4.2 void CSPCSDlg:: OnSelchangeBarcodeCombo ()

This function is called when a user selects a new barcode device from the Barcode device selection list. Let's assume your derived class is called CMyBarcodeDevice. Simply add the following code to the function.

```
CMyBarcodeDevice * mbd = new CMyBarcodeDevice (); // create an instance
if(str == mbd ->GetDeviceName())            // did the user select this device?
  APP->barcodeDevice = mrd;                 // if yes assign it to the global variable
else
  delete mrd;                               // else delete the instance
```

2.4.3 devices.h

There is still one last thing to do to add support for a new barcode device. Open devices.h and include the .h file for your derived class.

3. User Interface and Processing Thread Synchronization

There are two threads running as part of the SPCS program. These threads are the User Interface thread and the Processing Thread. The following sections describe how the threads synchronize.

3.1 Start and Stop

Figure 3-1 shows how events are used to synchronize starting and stopping. When the user clicks the start button, the user interface signals startEvent. This tells the processing thread to begin the label writing process.

When the user clicks stop, the user interface signals stopEvent then waits for the processing thread to signal either finishedEvent or threadKilled.

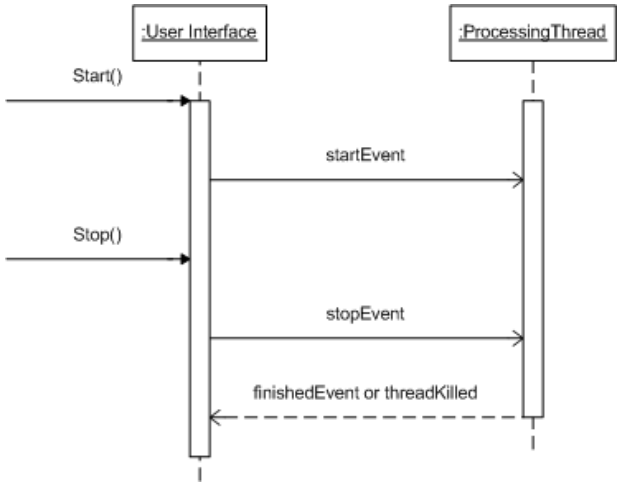

Figure 3-1 Start and Stop

©Metalcraft, Inc., 2002

3.2 Exiting the Application

Figure 3-2 shows the events for exiting the application. When the user chooses to exit the application the user interface firsts checks if the processing thread is running. If the processing thread is running the user interface stops it by signaling stopEvent. The user interface signals killEvent which notifies the processing thread to exit then restarts the processing thread by signaling startEvent. Just before the processing thread exits, it signals threadKilled which tells the user interface it can now exit.

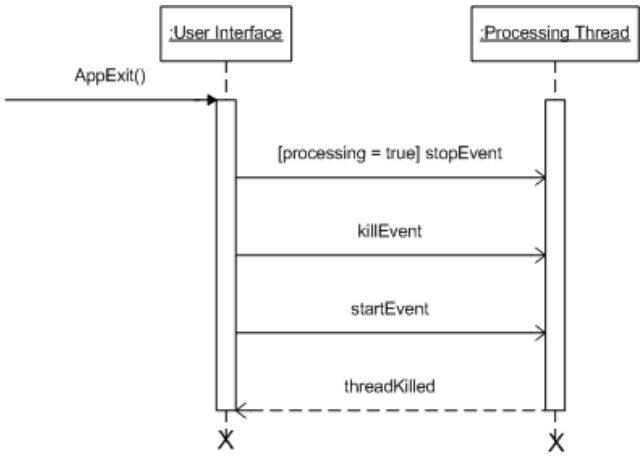

Figure 3-2 Exiting the Application

3.3 Thread Died

When the processing thread incurs a fatal error and must exit, it sends a WM_THREAD DIED message to the user interface thread. The user interface displays the error then exits. See Figure 3-3

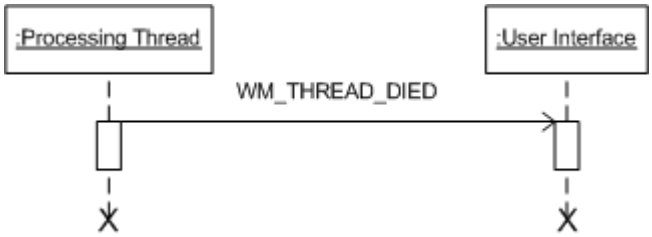

Figure 3-3 Thread Died

4. Processing Thread and Web Handler Synchronization

Figure 4-1 shows the signal handshaking that occurs between the processing thread of the SPCS and the web handler unit.

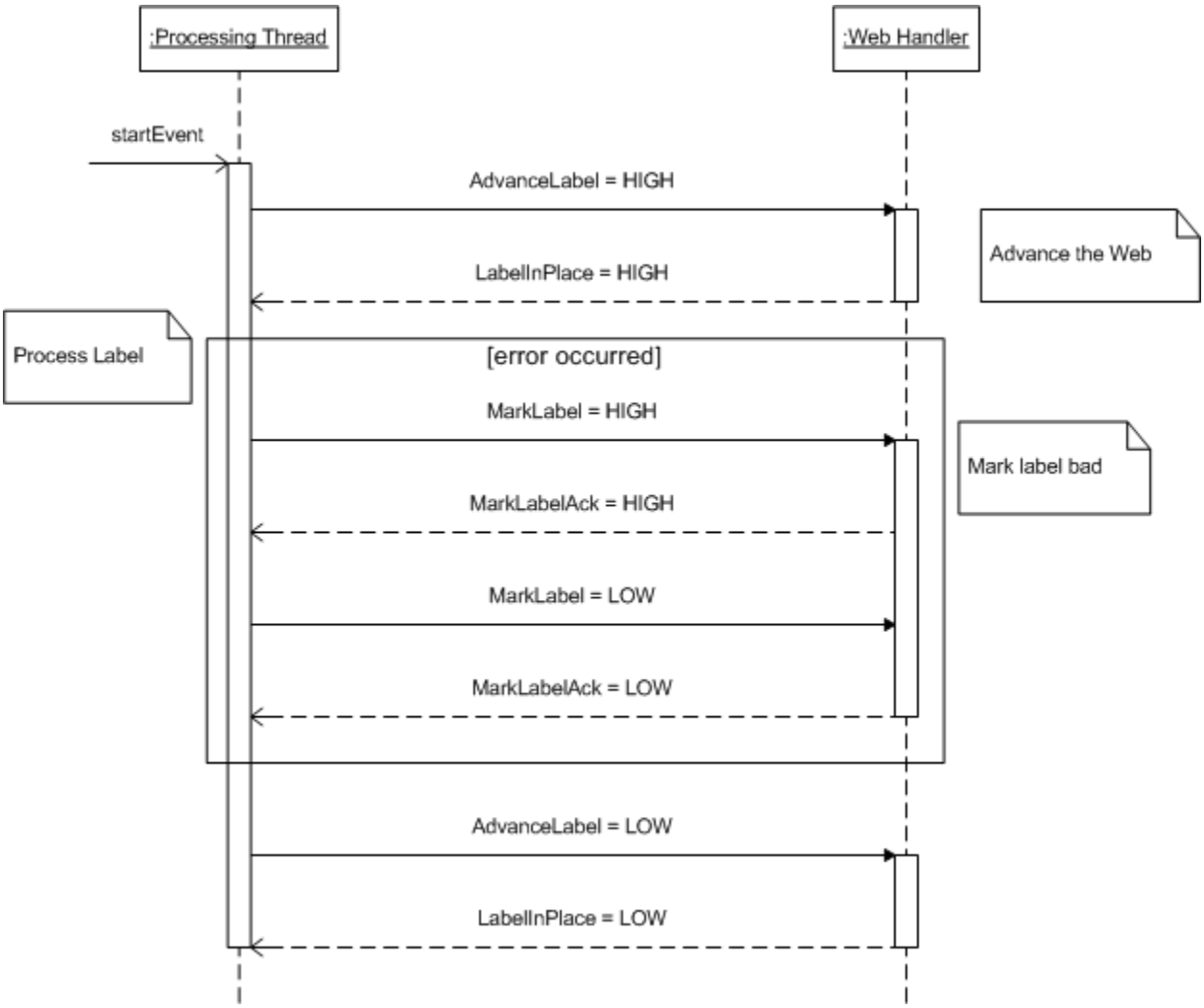

Figure 4-1 Processing Thread and Web Handler Signals

5. Port Wiring

Figure 5-1 shows the how the cable between the PC and the web handler is wired. The parallel port on the PC did not have enough power for the application so an external power source and pull-up resistors were added to boost the signal.

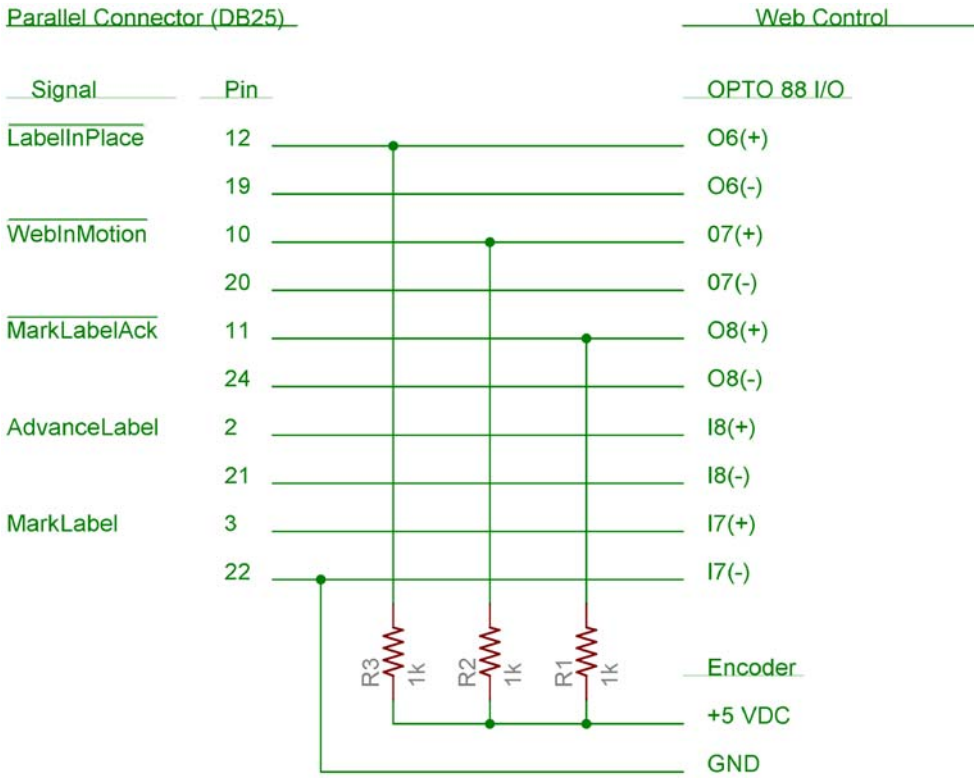

Figure 5-1 Port Wiring Diagram

6. File Formats

6.1 RFID Image Database

6.1.1 Types

Items are stored in the RFID Image Database based on the standard types in the IA32. Specifically the following types are used.

UINT32 – A little endian, 32-bit, unsigned integer
DOUBLE – A 64-bit double precision floating-point number (Little endian)
STRING – A null terminated ASCII character string

6.1.2 File Description

The first four bytes in the file contain the number of records in the database stored as a UINT32. Next follows a record descriptor for each record in the database.

<UINT32>Number of Records | RECORD | RECORD | …

6.1.3 Record Descriptor

The first four bytes of the record descriptor contain the size of the record stored as a UINT32. The record size includes the four bytes that store the size. Next comes the barcode data associated with the record stored as a STRING. If no barcode is associated with the record, the string consists of a single null character. Next comes the number of buffers in the record stored as a UINT32. Next comes a buffer descriptor for each buffer in the record.

<UINT32>record size | <STRING>barcodeData | <UINT32>numberOfBuffers | BUFFER | BUFFER | …

6.1.4 Buffer Descriptor

The first four bytes of the buffer descriptor contain the start address for the buffer stored as a UINT32. Next comes the size of the buffer in bytes stored as a UINT32. Note this is the size of the buffer not the size of the buffer descriptor. Last comes the actual buffer data.

<UINT32>start address | <UINT32>bufferSize | bufferdata

©Metalcraft, Inc., 2002

6.2 Save Settings

6.2.1 Global variables

First the global variables are saved. They are four bytes each and saved in the following order: backToBackErrors, runType, totalErrors and totalLabelCount.

6.2.2 Image Database Path and Filename

Next the image database path and filename are saved as a null terminated ASCII string. For example:

```
C:\My Documents\test.rdb
```

If no database was loaded, only a null character is saved.

6.2.3 Barcode Device Identifier

Next a description of the barcode device is saved. This description must match the description returned by GetDeviceName exactly in order for the SPCS program to load the correct device.

6.2.4 RFID Device Identifier

Next a description of the RFID device is saved. This description must match the description returned by GetDeviceName exactly in order for the SPCS program to load the correct device.

6.2.5 Barcode Eye Catcher

Next a barcode eye catcher is saved. This eye catcher is the four ASCII characters “BARD”. This is useful when viewing the file with a hex editor.

6.2.6 Barcode Settings

Next the settings for the barcode device are saved. They are different for each barcode device but usually consist of at least a port number.

6.2.7 RFID Eye Catcher

Next a RFID eye catcher is saved. This eye catcher is the four ASCII characters “RFID”. This is useful when viewing the file with a hex editor.

6.2.8 RFID device settings

Lastly, the settings for the RFID device are saved. They are different for each RFID device, but usually consist of at least a port number.